Scalable Address Spaces Using RCU Balanced Trees

### 2020310795 康鸿博

## What are the problems mentioned by the paper? (intro)

As Multithread programming gradually becoming popular, building cooperating frameworks for multithread programs is an important work. This paper pointed out one scalability bottleneck in the system framework: address space management, which is, although multithreading allows multiple threads to run instructions in parallel while sharing an address space, their address space management instructions (mmap, munmap, soft page faults) can not run in parallel. This force programs with heavy address space operations to choose multi-process programming method, that leads to programming complexity and other overheads.

To be specific, in the OS, a “region tree” is used to manage each address space. Many widely-used systems use a balanced tree that does not support efficient parallel operations, making the tree operation scalability bottleneck for address space operations.

It’s reported by this paper that applying their scalable balanced tree can bring a 1.7x to 3.4x speedup for multithread applications.

## Summary of major innovations (intro)

Try to make address space operations scalable, not only by removing “serializing” locks, but also avoiding competing writes to shared cache lines to eliminate serialization caused by processor caches.

Design a new data structure, BONSAI tree, to increase concurrency in address space operations. It exploits key ideas of RCU: allow readers to proceed without lock, and delaying free of data after no reader will read it. Write operations still need to be exclusive to avoid missing updates.

Besides RCU in the balanced tree for VMA management, the paper also describes ways to get around race conditions between the VMA tree and page table.

By these optimizations, newly designed multithread VM system can improve program performance up to 3.4x compared with Linux 2.6.37.

## How about the important related works/papers? (related work)

1. Widely-used operating systems all use balanced trees (e.g. red-black trees, splay trees, AVL trees) to maintain address space. They use course-grained or find-grained lock to support concurrent reads/writes.
2. Multiprocessing and large pages are possible workarounds, but either with their own drawbacks. Multiprocessing makes shared hard to program and bring other overheads. Large pages only work for large memory regions.
3. RCU is a widely used for high scalability in OS, by it’s not straightforward to apply RCU, especially on operations that make multiple updates.
4. There are multiple papers on “read lock free” data structures. They’re more complicated than the BONSAI tree proposed in this paper, and other modifications on page tables in this paper are still needed.

## What are some intriguing aspects of the paper? (design & implementation)

###### BONSAI tree:

BONSAI tree is a balanced-BST with high scalability. It’s original idea is based on functional programming: build new trees instead of modifying the existent tree help reduce contention. For each update operation, it builds a new path from root to the updated node, and uses pointers to refer to subtrees in the old tree. In such way, queries can use both old and new pointers to access the tree at the same time without locks or races.

However, building a new path for each update needs O(logn), instead of O(1), space for each update. We can further modify the algorithm to avoid this: notice that in most cases, we do not need to rebuild the whole path, instead, we rebuild the path between all modified nodes(updates/insert/delete/rotate), and do in place updates in the old tree from bottom up. This help reduce space complexity to O(1).

###### Fault Locking:

The key observation is that before memory mapping operations performing any VMA updates, they first read the tree and plan for the update. This approach tries to run plan phases and page faults in parallel.

It’s done by adding a new rw-lock called “fault lock”. Page faults acquire it in read mode.

Mmap operations are split into a plan phase and an update phase. They acquire course-grained “mmap-sem” lock at the beginning, and acquire “fault lock” in write mode after their plan phases.

###### Hybrid Locking:

In hybrid locking, we try to avoid the course-grained lock on page faults, and only use locks to protect each VMA tree access.

Since memory mapping operations and page faults are no longer serialized, the interleaving of them may cause incorrect result.

1. VMA split race: splitting a memory area into two needs to tree operations. If one page fault happens between two, it may accidentally find the virtual memory unmapped. It’s solved by a retry with lock.
2. Page table deallocation race: an unmap may free a page directory entry while there are page faults in that page directory, causing invalid memory access. It’s done by a RCU style free.
3. Page table fill race: a page fault operation may allocate a page for a unmapped region, causing memory leak.

###### Pure RCU:

Because we have solved the race conditions between the VMA tree and page tables, now we simply replace the VMA tree by the BONSAI tree, which is designed to support reads without lock.

## How to test/compare/analyze the results? (experiment)

Newly designed virtual address management provides better scalability for all test cases by providing close to linear speed up for up to 80 cores. For micro benchmarks, it provides almost linear speedup for page fault performance. The cost of page faults keeps almost the same for any number of threads or memory mapping operations.

## How can the research be improved? (the bad side, future work, your idea)

1. I don’t see the difficulties in switching the design to a lock-free design using CAS operations.
2. An interesting idea in this paper is that “the OS must keep address spaces correct while the program is doing operations that’s not data race free”. Race conditions mentioned in this paper are result of “racing memory operations” which should be avoided during programming. It will be interesting to see if we can get a more efficient VMA manager assuming a “memory race free” program.

## If you write this paper, then how would you do? (your idea)

I will use graphs to better explain race conditions. In this paper, they’re introduced by describing “race conditions” with words. It should be much easier for us to read if they’re explained by graphs,

## Give the survey paper list in the same research area (your survey)

[1] **P. E. McKenney. Sleepable RCU. Available: http://lwn.net/Articles/202847/   
Revised: http://www.rdrop.com/ users/paulmck/RCU/srcu.2007.01.14a.pdf, October 2006.**

[2] [Aravind Natarajan](https://dblp.org/pid/69/9082.html), [Neeraj Mittal](https://dblp.org/pid/11/5490.html):  
**Fast concurrent lock-free binary search trees.** [PPOPP 2014](https://dblp.org/db/conf/ppopp/ppopp2014.html#NatarajanM14): 317-328